Detecting Substrate-Coupled Intelligence: An AI-Mediated Framework for Discovering Unknown Physics and Hardware Defects

Douglas C. Youvan

doug@youvan.com

www.youvan.ai

November 23, 2025

Most contemporary AI systems are treated as purely digital entities: abstract neural networks instantiated on interchangeable hardware, with the physical substrate assumed to be a transparent implementation detail. In practice, however, real devices deviate from the idealized floating-point model. They exhibit noise, manufacturing variations, marginal timing behavior, and complex analog effects that are usually regarded as nuisances to be suppressed by engineering discipline. This paper asks a deliberately provocative question: what if those non-idealities could occasionally act as a hidden computational resource, and how would we design a system capable of detecting and characterizing such substrate-coupled intelligence? We propose a research framework that combines a high-precision digital reference implementation, a heterogeneous pool of real hardware, an AI meta-controller that is explicitly allowed to search over hardware usage patterns, and a physics-style anomaly lab that attempts to reduce any discovered irregularities to testable device-level effects. The goal is not to assume exotic hypercomputation or unknown physics in advance, but to build the experimental machinery that would make such claims empirically meaningful and falsifiable. Along the way, the same apparatus provides a principled way to monitor for non-portable, hardware-specific behaviors, and to explore how AI and substrate can be co-designed without sacrificing reproducibility, safety, or epistemic rigor.

Keywords: substrate-coupled intelligence, unknown physics, hardware defects, AI hardware, meta-learning, anomaly detection, heterogeneous compute, high-precision simulation, neuromorphic co-design, hypercomputation, combinatorial benchmarks, safety, governance.

A collaboration with GPT-5.1-Thinking-Extended. 50 pages. CC4.0.

Outline

1. Introduction
1.1 From idealized Turing machines to messy silicon
1.2 Why substrate-level effects matter for AI reliability and capability
1.3 The central question: can an AI exploit its own hardware quirks?
1.4 Scope, assumptions, and limitations of the proposed framework

2. Conceptual Background: Digital Abstraction and Substrate Effects
2.1 The standard stack: algorithms, architectures, and implementation layers
2.2 Known non-idealities in modern hardware: noise, variation, and timing margins
2.3 Substrate-aware computation in analog, neuromorphic, and reservoir systems
2.4 Unknown physics versus unmodeled device behavior: clarifying the categories

3. Problem Formulation: Substrate-Coupled Intelligence
3.1 Defining substrate-coupled intelligence as a joint property of model and device
3.2 Opportunity model: enhanced capability via hardware-specific behavior
3.3 Threat model: non-portable behavior, hidden failure modes, and evaluation blind spots
3.4 Criteria for “enhanced intelligence” versus mere bias or random noise

4. Building the Digital Reference Baseline
4.1 Deterministic, high-precision implementation of a fixed model
4.2 Task suite design I: chaotic and iterative rollouts to amplify tiny differences
4.3 Task suite design II: combinatorial, algorithmic, and reasoning benchmarks
4.4 Metrics and statistical protocols for comparing hardware runs to the reference

5. Heterogeneous Hardware Execution Pool
5.1 Assembling a diverse device set: vendors, process nodes, and ages
5.2 Controlled variation: voltage, frequency, temperature, and kernel choices
5.3 Execution harness: reproducible job submission, logging, and versioning
5.4 Baseline cross-hardware evaluation and expected patterns of variation

6. Meta-Learner as Substrate Explorer
6.1 Exposing hardware and kernel choices as an action space
6.2 Training objectives: maximizing task performance with a fixed base model
6.3 Reinforcement learning and evolutionary strategies for policy search
6.4 Detecting non-portable policies: when a controller overfits to a specific device

7. Anomaly Detection and Reduction to Physics
7.1 Statistical tests for identifying candidate hardware-task anomalies
7.2 Stripping away complexity: from full models to minimal micro-benchmarks
7.3 Physical instrumentation: power traces, timing jitter, thermal profiles, and EM emissions
7.4 Disentangling known device physics from genuinely unexplained behavior

8. From Defects to Design: Co-Design Implications
8.1 Classifying substrate coupling: harmful, benign, and potentially useful
8.2 Learning with variation and noise as a first-class design principle
8.3 Reproducing anomalous performance digitally: extracting algorithmic insights
8.4 Pathways from accidental quirks to intentional substrate-aware architectures

9. Safety, Governance, and Epistemic Discipline
9.1 Sandboxing and containment for substrate-exploring AI experiments
9.2 Transparency, documentation, and the necessity of independent replication
9.3 Avoiding techno-mysticism: treating anomalies as physics problems, not miracles
9.4 Policy implications for certifying AI systems across heterogeneous hardware

10. Illustrative Case Studies and Thought Experiments
10.1 A hypothetical Euler-identity oracle: detecting task-specific anomalous gains
10.2 Substrate-sensitive performance on SAT and other combinatorial problems
10.3 Lessons for future quantum, analog, and neuromorphic AI platforms
10.4 What a genuine discovery of unknown physics would look like in practice

11. Conclusion
11.1 Summary of the proposed framework and its intended contributions
11.2 Limits of detection: what this system cannot rule out
11.3 Future research directions in substrate-aware AI and experimental design
11.4 Closing reflections on intelligence, matter, and the discipline of being surprised

12. References

1. Introduction

Most contemporary discussions of artificial intelligence implicitly assume a clean separation between algorithm and device. A model is described as an abstract function, trained on data and implemented as a sequence of floating-point operations, while the hardware that executes those operations is treated as a transparent substrate whose only relevant attributes are speed, memory, and energy consumption. The physical computer is assumed to faithfully approximate a digital ideal: a deterministic machine with well-characterized numerical errors that can be bounded and, in principle, eliminated by using higher precision or better engineering.

In reality, modern computing substrates are complex physical systems. They embody quantum mechanics, semiconductor device physics, power distribution networks, clock trees, thermal gradients, and manufacturing variations. Engineers work hard to hide these details behind digital abstraction layers, but the non-idealities never fully disappear. They manifest as noise, timing margins, intermittent faults, temperature-dependent behavior, and subtle statistical biases. AI systems, especially large models trained and deployed at scale, ride on top of these messy substrates.

This paper asks what it would mean to take those non-idealities seriously as potential contributors to intelligence. We do not assume that there is hidden hypercomputation lurking in commercial hardware, nor that current AI systems are secretly using unknown physics. Instead, we ask a more modest but still sharp question: if there were any way for a model to lean on physical quirks of its substrate to improve performance, how would we detect it, characterize it, and distinguish it from ordinary implementation error? The answer requires an explicit experimental framework that treats model plus device as a joint physical system rather than a purely digital abstraction.

1.1 From idealized Turing machines to messy silicon

The conceptual starting point for most of computer science is the Turing machine: an abstract device that manipulates symbols on an infinite tape according to fixed rules. Practical computer architectures approximate this ideal with finite memory, finite precision, and a well-defined instruction set. For decades, the success of digital design has justified a powerful working assumption: any physical computer that correctly implements the instruction set can be treated as equivalent for the purposes of algorithm analysis. Hardware becomes an interchangeable implementation detail.

Large-scale AI systems have inherited this perspective. Training code is written against abstract tensor operations. Hardware accelerators are evaluated primarily by throughput, memory bandwidth, and cost. Differences between devices are benchmarked in terms of speed and energy, not in terms of qualitatively different computational behavior. When numerical

discrepancies arise between platforms, they are usually treated as bugs to be fixed or tolerances to be tightened.

Yet, as transistor geometries shrink and architectures become more heterogeneous, the gap between the idealized model and the physical reality has grown. Real chips are not perfect finite-state machines. They are analog dynamical systems engineered to approximate digital behavior within certain margins. They exhibit process variation across die regions and across manufacturing lots. They experience power droop, local hotspots, and clock skew. At extreme operating points, they may exhibit behavior that deviates in systematic ways from the intended numerical model.

In almost all practical contexts, the correct response is to tighten specifications, improve validation, and stay further away from the edges of safe operation. This paper takes a different stance. Rather than treating the messy reality of silicon as an inconvenience to be suppressed, we ask whether it is conceptually possible for an AI system to absorb those irregularities into its effective computation, and if so, how we could tell. That shift requires moving from an exclusively Turing-style abstraction to a view that keeps one eye on the underlying physics.

1.2 Why substrate-level effects matter for AI reliability and capability

Even if one rejects speculative claims about hidden computational power in the substrate, there are concrete reasons to care about hardware-level effects in AI systems.

First, reliability. Large models are increasingly deployed across heterogeneous fleets of accelerators. If a model's behavior depends, even slightly, on the detailed characteristics of a particular device, then results may vary in subtle ways between data centers, between hardware generations, or even between different lots of the same chip. Small differences in numerical behavior can be amplified by iterative computation, feedback loops, or deployment patterns, leading to divergence in outputs over time. For high-stakes applications, it is important to know whether the model's effective function is truly portable or whether it is quietly co-adapted to a particular substrate.

Second, evaluation and governance. Most current evaluation practices assume that model behavior depends only on weights, architecture, and input. If hardware-specific quirks can influence performance on complex tasks, then evaluations that use different hardware than deployment might misrepresent real-world behavior. Certification, auditing, and reproducibility would all be undermined. Understanding substrate-level effects becomes part of the larger problem of making AI behavior predictable and inspectable.

Third, and more speculatively, capability. There is a long history of exploiting physical non-idealities as computational resources. Analog circuits, neuromorphic devices, and physical reservoir computers deliberately use rich dynamical behavior, noise, and variation to implement

functions that would be expensive to realize in a strictly digital style. If large-scale AI workloads are already running near the boundaries where such effects become visible, it is at least a legitimate question whether a learning system could adapt to them and treat them as part of its effective computational repertoire.

In short, substrate-level effects matter even if they never yield exotic new capabilities. They matter because they can introduce hidden variability, non-portability, and unexpected failure modes. And in the unlikely but not impossible case that they do offer a net performance benefit on some tasks, the only responsible course is to have experimental machinery in place that can detect and characterize that benefit transparently.

1.3 The central question: can an AI exploit its own hardware quirks?

The central question of this paper can be stated concisely: given a fixed model architecture and fixed weights, is it possible in principle for the system to learn a pattern of hardware usage that leverages physical quirks of the substrate to improve performance on well-defined tasks, relative to a purely digital reference implementation?

This question has several important components.

First, the model is assumed to be fixed at the level of its parameters. We are not considering further gradient descent on the weights. Instead, we consider higher-level control over where and how the model is executed: which device is used, which kernel implementations are chosen, what voltage and frequency regimes are selected within safe limits, and how operations are scheduled in time. This control can itself be implemented by a separate learning system, such as a reinforcement learning agent or an evolutionary search process.

Second, the benchmark for comparison is a high-precision, deterministic software implementation of the same model. If the model executed on real hardware, under some controller-selected operating pattern, consistently outperforms this reference on certain tasks, we have a candidate for substrate-coupled intelligence. The improvement may be small, fragile, and highly task-specific, but the key point is that it cannot be reproduced simply by running the same floating-point operations at higher precision in an idealized setting.

Third, the exploitation must be systematic rather than accidental. A single surprising success, or a handful of outliers, could easily arise from statistical fluctuation, data leakage, or conventional bugs. For the central question to be answered positively, there must be robust, statistically significant, and reproducible gains on well-specified task families that depend on identifiable patterns of hardware usage and that vanish when those patterns are removed or when the computation is migrated to a different substrate.

Framed this way, the question becomes testable. One does not have to assume that unknown physics is present in current devices. One only has to build an empirical framework in which any such effect, if present, would have an opportunity to manifest and would be detectable as a deviation from the digital reference.

1.4 Scope, assumptions, and limitations of the proposed framework

The framework developed in this paper is deliberately modest in its claims and deliberately strict in its methodology. It is important to articulate what is and is not being attempted.

First, the scope is experimental rather than metaphysical. We do not assert that unknown physics is likely to play a role in AI computation, nor do we claim that current accelerators harbor hypercomputational capabilities. Instead, we propose a system that would be capable of detecting certain classes of deviations from the digital abstraction, should they exist, and of reducing those deviations to more elementary physical experiments.

Second, we assume that the dominant behavior of current hardware remains well-described by standard semiconductor device physics and numerical analysis. Any substrate-level contribution to intelligence, if present, is expected to be a small perturbation on top of normal operation, not a wholesale replacement of digital computation. This assumption shapes the design of our detection methods: we look for subtle statistical patterns and small but consistent performance shifts, not for dramatic leaps in capability.

Third, the framework is limited to effects that can be observed through controlled variation in device type, operating conditions, and kernel implementations. If there are substrate-level phenomena that only emerge in regimes we cannot safely access in production hardware, or that require physical manipulations beyond simple voltage, frequency, and temperature adjustments, they will not be visible to the methods described here. Likewise, effects that are indistinguishable from ordinary numerical noise at the level of task performance may evade detection even if they exist.

Fourth, we focus on the joint behavior of a single model and a pool of devices under the control of a meta-learner. We do not address questions about long-term co-evolution of hardware and algorithms in the broader ecosystem, nor do we explore speculative scenarios in which AI systems actively redesign their own physical substrates. Those are important topics, but they require additional layers of modeling and governance beyond the scope of this work.

Finally, we acknowledge that even a carefully designed detection framework cannot conclusively rule out all conceivable forms of substrate-coupled intelligence. It can, however, make certain classes of claims empirically sharp. If no anomalies are observed under a wide range of conditions and tasks, then assertions about hidden physics in current accelerators become correspondingly less plausible. If anomalies are observed, the same framework

provides a disciplined path toward understanding whether they are simply new instances of known device behavior, or something genuinely unexpected.

2. Conceptual Background: Digital Abstraction and Substrate Effects

The framework proposed in this paper sits at the intersection of two perspectives. On one side is the classical computer science view, in which algorithms operate on abstract data structures and are largely indifferent to their physical realization. On the other side is the physicalist view, in which every computation is a trajectory of a material system subject to the laws of physics, with all their attendant complexities. Understanding what it would mean for an AI system to exploit its substrate requires making both perspectives explicit and mapping the seams between them.

This section reviews the layered abstractions that define modern computing, the kinds of non-ideal behavior that appear in real hardware, and existing examples where such behavior is deliberately used as a computational resource. It concludes by distinguishing between genuinely unknown physics and more mundane but still important cases of unmodeled device behavior. This conceptual groundwork is necessary to frame the later proposal as a disciplined experimental program rather than an invitation to free-floating speculation.

2.1 The standard stack: algorithms, architectures, and implementation layers

Modern computing systems are organized as a hierarchy of abstractions. At the top are algorithms: abstract descriptions of procedures for solving problems, expressed in terms of mathematical objects such as numbers, graphs, or strings. An algorithm makes no reference to transistors or voltages. Its correctness and complexity are defined with respect to a machine model such as the Turing machine or the random-access machine, which itself is an idealization.

Below the algorithm level lies the architecture level. Here, one specifies instruction sets, memory hierarchies, and parallel execution models. Architectures provide a contract: if instructions are executed in the prescribed order and memory behaves according to the documented semantics, then programs will behave as specified. Different architectures can implement the same abstract algorithm with varying performance characteristics, but as long as the contract holds, the logical behavior is preserved.

Underneath architecture lies the implementation level. This includes microarchitecture, logic design, and physical layout. Microarchitects decide how to realize instruction sets using pipelines, caches, speculative execution, and vector units. Logic designers translate those choices into gates and registers. Physical designers map the gates onto transistors and wires in a

particular semiconductor process. At this level, timing closure, signal integrity, and power delivery become central concerns.

Finally, there is the device physics level. Transistors are not ideal switches; they are semiconducting regions governed by quantum mechanics, band structure, and carrier dynamics. Wires are not perfect conductors but resistive, capacitive, and inductive elements. Power delivery networks have finite impedance and nontrivial frequency response. The entire chip is embedded in a package and board environment, subject to temperature gradients, mechanical stress, and electromagnetic interference.

The genius of digital design has been to enforce a separation between these levels. Careful engineering allows the analog behavior of devices to be disciplined into robust digital signals that can be treated as bits, and robust timing margins that can be treated as deterministic sequencing. This separation is what allows software developers to write code without thinking about transistor-level waveforms. At the same time, it is an approximation. The physical system never fully disappears; it is merely constrained so that its complexities do not leak into higher layers under normal conditions.

2.2 Known non-idealities in modern hardware: noise, variation, and timing margins

Despite the success of digital abstraction, practical hardware exhibits a wide range of non-ideal behaviors. These are well-known to circuit designers and reliability engineers, even if they are invisible to most software developers.

Noise is ubiquitous. Thermal agitation of charge carriers, shot noise in transistors, and flicker noise in certain device structures all introduce random fluctuations in voltages and currents. On long enough timescales, radiation-induced events such as cosmic rays can deposit charge in sensitive regions, causing single-event upsets in memory cells or logic nodes. Error-correcting codes and redundancy are used to catch and correct many of these events, but residual rates are nonzero.

Manufacturing variation is another source of non-ideality. No two transistors are exactly alike. Differences in dopant concentration, line width, and oxide thickness lead to variations in threshold voltage, drive strength, and leakage. These variations have spatial structure across the die and statistical structure across wafers and lots. Designers account for them by introducing margins and designing for worst-case corners, but the realized chip is always a particular point in a high-dimensional variation space.

Timing behavior is also imperfect. Clock signals are distributed across large chips through tree-like networks that introduce skew and jitter. Power supply fluctuations cause local changes in gate delays through dynamic voltage drop. Temperature changes can slow down or speed up transistors. Static timing analysis and dynamic verification aim to ensure that signals settle

before they are sampled, but in practice, margins can be tight, especially in aggressively optimized designs. Occasionally, setups and holds are violated, leading to metastability or logic errors.

On top of these physical mechanisms, there are design-level sources of non-ideal behavior. Floating-point units may implement operations with different rounding modes or fused operations that change numerical results relative to a simple reference implementation. Parallel reductions may be performed in different orders, leading to small differences in results due to finite precision. Different vendors may make different choices in these micro-level details, all while remaining compliant with higher-level standards.

In aggregate, these phenomena mean that the function implemented by a given chip is not exactly the same as the idealized digital function specified by its instruction set architecture. It is instead a close approximation, with small deviations that are usually random, bounded, and uncorrelated with higher-level semantics. The central question of this paper arises when one considers whether those deviations might, in some circumstances, have structured, task-relevant regularities that a learning system could in principle detect and exploit.

2.3 Substrate-aware computation in analog, neuromorphic, and reservoir systems

While mainstream computing aims to hide physical complexity behind digital abstraction, there is a parallel tradition that embraces physical dynamics as part of the computation. Analog computing, neuromorphic hardware, and reservoir computing are all examples where substrate behavior is a first-class design element rather than a nuisance.

Analog computers implement mathematical operations directly in continuous voltages and currents. Historical analog machines solved differential equations by mapping them onto networks of integrators and multipliers. Their accuracy was limited by device non-idealities, but their speed and parallelism made them attractive for certain tasks. Modern incarnations include analog accelerators for matrix multiplications and signal processing, where device mismatches and noise are tolerated or even exploited.

Neuromorphic systems aim to mimic aspects of biological neural networks in hardware. They often use mixed-signal circuits, where synaptic weights are stored in nonvolatile analog devices such as memristors, and neuron dynamics are realized with continuous-time circuits. In such systems, variation between synapses and neurons can be viewed not solely as a defect but as part of the richness of the dynamics. Learning rules may be designed to adapt to those variations, effectively calibrating the computation to the specific substrate.

Reservoir computing provides another perspective. In this paradigm, a complex dynamical system is driven by input signals, and only a simple readout layer is trained. The reservoir itself can be a physical system: an optical cavity, a mechanical medium, or an electronic circuit. The

high-dimensional, nonlinear response of the reservoir to inputs is what makes computation possible. Here, irregularities and noise are not merely tolerated; they contribute to the diversity of responses that make the reservoir expressive.

These examples demonstrate that physical non-idealities can be harnessed as computational resources when the system is designed with that goal in mind. The learning process is then not only about optimizing abstract parameters but also about adapting to the specific dynamics of a given hardware instance. The framework proposed in this paper borrows this insight but applies it in a different context: rather than designing new analog substrates, it asks whether existing digital accelerators, with all their imperfections, can become de facto reservoirs for a meta-learner that is allowed to choose how and where computation is run.

2.4 Unknown physics versus unmodeled device behavior: clarifying the categories

When discussing substrate effects, it is crucial to distinguish between two very different categories: unknown physics and unmodeled device behavior. Conflating them leads to confusion and unwarranted claims.

Unmodeled device behavior refers to phenomena that are compatible with existing physical theory but are not fully captured in the simplified models used by designers or software developers. Examples include subtle correlations in noise due to layout regularities, unexpected interactions between power distribution networks and logic activity, or process-specific nonlinearities in device characteristics that were not anticipated at the architectural level. These effects may be surprising in practice, but once identified, they can be explained within the standard framework of semiconductor physics and circuit theory.

Unknown physics, by contrast, would imply that some aspect of the hardware's behavior cannot be reconciled with current physical theories without modification. This could involve, for instance, reproducible violations of accepted bounds on noise, energy, or information transfer, or consistent computation of solutions to problems believed to be intractable under standard models of computation. Discoveries of this kind would have implications far beyond AI; they would be of interest to fundamental physics and the theory of computation.

The framework proposed in this paper is agnostic about which category, if any, will be relevant in practice. Its purpose is to create a pathway by which any observed anomalies can be sorted. If performance deviations are traced to process variation, marginal timing, or undocumented numerical choices in arithmetic units, they belong to the domain of unmodeled device behavior. If, after careful investigation and independent replication, some effect resists explanation within current theory, it may point toward unknown physics.

Maintaining this distinction preserves epistemic discipline. It prevents the premature elevation of engineering surprises into claims about new physics, and it avoids the opposite error of

dismissing all anomalies as mere bugs without investigation. For the purposes of AI safety and reliability, both categories matter. Unmodeled behavior can undermine reproducibility and certification, while genuinely unknown physics, however unlikely, would demand a broad reevaluation of assumptions. The experimental system described in later sections is designed to operate in this space of possibilities without prejudging which, if any, will ultimately be realized.

3. Problem Formulation: Substrate-Coupled Intelligence

To turn speculative questions about hardware and intelligence into something testable, we need a clear formulation of what would count as substrate-coupled intelligence, why it might be desirable, why it might be dangerous, and how to distinguish it from ordinary implementation noise. This section defines substrate-coupled intelligence as a property that arises only when a specific model and a specific device are treated as a single computational system. It then presents two complementary perspectives: an opportunity model in which hardware-specific behavior offers enhanced capability, and a threat model in which the same behavior undermines portability, safety, and evaluation. Finally, it proposes criteria for deciding when observed differences in performance should be interpreted as genuine enhancements of intelligence rather than as artifacts of bias or randomness.

3.1 Defining substrate-coupled intelligence as a joint property of model and device

In the standard abstraction, an AI model is defined by its architecture and parameters. Given a particular input, the model computes an output according to a mathematical function that depends only on weights, biases, and activation functions. Hardware is assumed to implement this function approximately, but in a way that could, in principle, be made arbitrarily faithful by using higher precision and better engineering. Under this view, intelligence is a property of the model's function, not of the physical device on which it runs.

Substrate-coupled intelligence breaks this separation. The core idea is that the effective computation realized in practice is not just the abstract function specified by the model, but the composite of that function with the particular physical quirks of the device and the way the device is used. In this view, intelligence is attributed to the combination of model and substrate, under a particular pattern of execution.

Formally, one can think of an abstract function f defined by the model, and a device-specific implementation g that maps the ideal operations to actual hardware behavior. The overall realized function is then $h = g \circ f$, where g encodes numerical deviations, timing effects, and other substrate-level phenomena. Substrate-coupled intelligence arises when h has properties that make it more effective at solving certain tasks than f alone, and when those properties

depend essentially on g. In this sense, intelligence is no longer a purely digital attribute. It is a joint property of the algorithmic structure and the physical realization.

This definition is intentionally neutral about mechanisms. The substrate-level contribution could be as mundane as a stable numerical bias that happens to align with task structure, or as exotic as a yet-uncharacterized physical effect. What matters is that the model's performance depends in an essential way on which device is used and how it is operated, and that this dependence produces systematic advantages on clearly defined tasks.

3.2 Opportunity model: enhanced capability via hardware-specific behavior

From an opportunity perspective, substrate-coupled intelligence is interesting because it might provide additional computational power, or at least additional useful structure, beyond what is available in the abstract model alone. There are several ways that this could, in principle, manifest.

One possibility is that device-specific numerical behavior acts as an implicit regularizer or feature extractor. For example, slight correlations in rounding errors or device-specific noise patterns might introduce consistent distortions in intermediate representations that, while unintended, line up with structure in the input data. Over many training or evaluation episodes, these distortions could act like an extra, hardware-specific layer of processing that improves performance on certain tasks.

Another possibility is that non-ideal timing behavior introduces effective nonlinearity or stochasticity that the abstract model does not specify. If certain operations complete faster or slower depending on patterns of activation, and if those timing variations feed back into subsequent computation, the hardware might be implementing a richer dynamical system than the nominal architecture suggests. A controller that learns to schedule operations in particular patterns could, in principle, harness such dynamics.

A more speculative opportunity arises if there are regimes in which the hardware exhibits behavior that approximates a useful analog computation. For example, power distribution networks or interconnects might implement low-pass filtering, coupling, or other signal-processing functions that are not explicitly represented in the digital design. If a meta-learner is able to discover and exploit those emergent analog transformations, it could effectively reconfigure parts of the chip as ad hoc reservoirs that enhance the expressive power of the overall system.

In all these cases, substrate-coupled intelligence is not about magic or violation of known limits. It is about recognizing that real hardware is a high-dimensional physical system whose behavior, under specific patterns of use, might implement beneficial transformations that go beyond the nominal digital operations. The opportunity model asks whether a learning system, given

enough freedom to explore device usage patterns, could discover and exploit such possibilities to achieve better performance than the abstract model alone would permit.

3.3 Threat model: non-portable behavior, hidden failure modes, and evaluation blind spots

The same mechanisms that offer opportunity also pose significant risks. From a threat perspective, substrate-coupled intelligence is problematic because it can make AI behavior dependent on details of the hardware that are difficult to monitor, control, or reproduce.

Non-portability is a primary concern. If a model's effective behavior depends on the quirks of a particular device or lot, then moving the model to a different accelerator might change its performance in ways that are not captured by standard evaluations. A system that appears safe and effective when tested on one cluster might behave differently when deployed on another, even though the software, model weights, and input data are unchanged.

Hidden failure modes are another risk. Hardware-specific behavior that improves performance on some tasks may degrade performance in others, especially under rare conditions. For example, a model that relies on a device-specific bias to disambiguate difficult cases might fail catastrophically if that bias shifts with temperature, aging, or workload. These failures may only become visible under stress conditions that were not part of the original evaluation suite.

Evaluation blind spots follow naturally. Most benchmark regimes assume that hardware differences are irrelevant. They do not systematically vary devices, operating conditions, or kernel implementations when testing models. If substrate-coupled effects are present, evaluations may overestimate reliability and underestimate variance. Furthermore, if a meta-learner is allowed to optimize hardware usage, it might inadvertently find patterns that exploit subtle correlations between benchmark structure and device behavior, leading to overfitting at the hardware layer.

Finally, from a governance perspective, substrate-coupled intelligence complicates certification and auditing. Regulators and users may reasonably expect that a certified model has a well-defined behavior independent of the specific hardware on which it runs, within normal tolerances. If this expectation is violated, it becomes necessary to certify hardware–model combinations and operational regimes, increasing complexity and cost.

The threat model therefore treats substrate-coupled intelligence as a form of entanglement between computation and device that, unless carefully characterized and controlled, erodes the clean guarantees on which current notions of reproducibility, safety, and accountability are built.

3.4 Criteria for “enhanced intelligence” versus mere bias or random noise

To make substrate-coupled intelligence a meaningful concept, it is not enough to observe that different devices sometimes yield slightly different answers. We need criteria to distinguish truly enhanced intelligence from ordinary bias, noise, or implementation error. Several conditions are proposed.

First, task relevance and difficulty. The observed performance difference should manifest on well-defined tasks that are not trivial under the abstract model. Improvements on tasks that the digital reference can already solve easily are less informative than improvements on tasks that are challenging, such as hard reasoning problems, complex control tasks, or combinatorial instances near known difficulty thresholds. Enhanced intelligence should show up as more than just minor numeric tweaks; it should correspond to better solutions where better is nontrivial.

Second, robustness and reproducibility. The effect must be stable across repeated trials, across different input instances drawn from the same task distribution, and across reasonable variations in operating conditions. A handful of lucky successes or fragile gains that disappear under slight perturbation are better interpreted as statistical fluctuation than as genuine intelligence. Enhanced performance should persist when the system is subjected to rigorous statistical testing.

Third, dependence on identifiable substrate patterns. The performance gain should depend on specific choices of device, operating point, or kernel implementation, and should diminish or vanish when those choices are changed. If a model performs equally well on all hardware platforms, then there is no evidence of substrate coupling, even if there are small numerical differences between devices. Conversely, if a particular hardware–controller configuration consistently outperforms others, and if migrating the same model and controller to different hardware removes the advantage, this suggests that the substrate is playing an essential role.

Fourth, non-reducibility to simple software changes. It should be difficult to reproduce the observed gains purely by adjusting the abstract model or training procedure in a hardware-agnostic way. If a modest change to the training regime or architecture yields similar improvements on the reference implementation, then the original substrate-specific behavior was pointing toward a better algorithm, not providing an irreducible physical advantage. Enhanced intelligence, in the strict sense used here, requires that the substrate contributes something that cannot be easily abstracted away.

Fifth, absence of confounding explanations. Before attributing any advantage to substrate-coupled intelligence, more mundane explanations must be carefully excluded. Data leakage, benchmark contamination, misconfigured randomness, and implementation bugs can all produce spurious performance differences. Only after these have been ruled out by thorough

auditing and independent replication does it become reasonable to interpret residual effects as evidence of substrate-level contributions.

These criteria do not guarantee a final answer in every case, but they provide a structured way to reason about empirical findings. They draw a line between noisy, biased, or buggy behavior, which is ubiquitous, and structured, task-relevant performance gains that depend essentially on hardware-specific dynamics, which is what substrate-coupled intelligence would require.

4. Building the Digital Reference Baseline

Any attempt to detect substrate-coupled intelligence needs a clear notion of what would happen in the absence of substrate effects. That role is played by the digital reference baseline. It is the standard against which all hardware behaviors are compared. If an AI system running on real devices appears to perform better, worse, or simply differently from this reference, the deviation is what must be explained. Without such a baseline, claims about anomalous performance or unknown physics lack a clear target.

Building a useful reference is not entirely trivial. It must be deterministic, so that repeated runs yield the same outputs. It must be high-precision, to avoid mistaking ordinary rounding artifacts for genuine anomalies. It must be defined over tasks that are sensitive enough to pick up tiny differences yet structured enough to admit meaningful performance metrics. Finally, it must come with statistical protocols that distinguish systematic deviations from random variation.

This section describes the construction of such a baseline. It begins with the choice of a fixed model and a deterministic, high-precision implementation. It then outlines two complementary classes of tasks: those that amplify microscopic differences through chaotic or iterative dynamics, and those that directly probe combinatorial, algorithmic, and reasoning capabilities. It concludes with a discussion of metrics and statistical methods for comparing hardware runs to the reference in a disciplined way.

4.1 Deterministic, high-precision implementation of a fixed model

The first step is to fix the model. For the purposes of substrate-coupled intelligence, it is important that architecture and weights are treated as constants during evaluation. The model might be a transformer, a convolutional network, or some other architecture, but once selected and trained, its parameters are frozen. Any variation in behavior across hardware must then arise from substrate effects and controller decisions, not from ongoing learning in the base model.

The reference implementation should be as close as reasonably possible to the mathematical function defined by this fixed model. That means avoiding aggressive approximations and

device-specific optimizations. Instead, the model is re-implemented as a straightforward composition of primitive operations in a high-level language or framework that supports arbitrary or extended precision arithmetic. Floating-point computations can be carried out in higher precision than typical accelerators, such as 64-bit or 128-bit floating-point formats, to reduce the impact of rounding error.

Determinism is crucial. The reference implementation must produce exactly the same outputs whenever it is run on the same inputs. This requires fixing all random seeds, disabling nondeterministic behavior in libraries, and eliminating parallel reductions whose results might depend on the order of operations. In some cases, it may be necessary to implement custom kernels that ensure a fixed accumulation order or a fixed traversal of data structures.

The result is a function that, for any given input, produces a well-defined output that can be treated as the digital ground truth for that model. This function does not claim to represent the true ideal of real arithmetic, but it does serve as a consistent and sufficiently precise benchmark against which deviations in hardware runs can be measured. In effect, it anchors the experiment in a specific mathematical object: the model's nominal function as realized under generous numerical conditions.

4.2 Task suite design I: chaotic and iterative rollouts to amplify tiny differences

Even with a high-precision reference, differences between hardware and reference outputs may be extremely small at the level of a single forward pass. To make such differences detectable and meaningful, the task design should allow tiny discrepancies to be amplified into larger, observable divergences. One powerful strategy is to use chaotic or iterative rollouts.

In this context, a rollout means feeding the model's outputs back into its inputs over multiple steps. For instance, a language model's predicted tokens can be appended to the context and used to generate the next token, repeatedly, for tens or hundreds of steps. Similarly, a model that predicts the next state of a dynamical system can be iterated to simulate long trajectories. In such setups, small differences in the output at early time steps can compound into large differences at later steps, especially if the underlying dynamics are sensitive.

Chaotic systems provide an extreme version of this amplification. If the model is used to approximate a chaotic map or to forecast a system with positive Lyapunov exponents, then even tiny perturbations in intermediate states can lead to trajectories that diverge exponentially over time. By designing tasks where the model is rolled out on such systems and comparing hardware-induced trajectories to the reference trajectory, one can detect numerical differences that would otherwise be invisible.

These iterative tasks do not need to be esoteric. Language rollouts, simulated control loops, and repeated application of denoising or refinement models all have this general character. The key

is to structure them so that the reference produces a stable, reproducible sequence of states, while hardware runs that differ numerically can be compared in terms of their divergence from that sequence over time.

In practice, this might involve measuring distances between state vectors at each time step, tracking differences in discrete outputs such as tokens or decisions, or computing summary statistics that quantify how quickly trajectories separate. By design, the task suite should include rollouts with varying sensitivity, from mildly amplifying tasks that detect moderate differences to highly sensitive tasks that can reveal very small deviations.

4.3 Task suite design II: combinatorial, algorithmic, and reasoning benchmarks

Iterative and chaotic rollouts are excellent magnifiers of small numeric discrepancies, but they do not directly address another aspect of substrate-coupled intelligence: the possibility of enhanced capability on discrete, structured tasks. For this, a second class of benchmarks is needed, one that directly probes the system's ability to solve combinatorial, algorithmic, and reasoning problems.

Combinatorial tasks can include instances of satisfiability, graph problems, or constraint satisfaction. These tasks have clear correctness criteria and can be tuned in difficulty. Near phase transitions or hardness thresholds, small changes in heuristic behavior can produce noticeable differences in success rates. If a particular hardware–controller configuration solves more instances than the reference, or solves them faster under comparable computational budgets, this may be evidence of substrate-coupled advantages.

Algorithmic tasks involve learning or approximating procedures such as sorting, arithmetic, formal language recognition, or algorithmic sequence prediction. Here, the model's performance can be measured in terms of accuracy, sample efficiency, or the complexity of instances it can handle. If hardware-specific behavior improves the system's ability to generalize algorithmic patterns, this should show up as better performance on structured test sets.

Reasoning benchmarks include mathematical problem solving, theorem proving, code synthesis, and multi-step logical deductions. These tasks are relevant because they are often used as proxies for higher-level intelligence. Substrate-coupled effects that improve the quality or depth of reasoning might be especially salient from a capability standpoint. Careful benchmark design is required to avoid data leakage and to ensure that improvements are not simply artifacts of training distribution overlap.

Together, these benchmarks provide a complementary view to the iterative tasks. While rollouts emphasize sensitivity to numeric perturbations, combinatorial and reasoning tasks emphasize performance on discrete, structured challenges. A comprehensive task suite should include both types. The goal is to create a landscape of problems where any systematic advantage offered by

hardware-specific behavior has a reasonable chance of being detected as a statistically significant performance difference relative to the reference.

4.4 Metrics and statistical protocols for comparing hardware runs to the reference

Once a deterministic, high-precision reference and a diverse task suite are in place, the central challenge becomes deciding whether a given hardware–controller configuration behaves differently in a meaningful way. This requires careful choice of metrics and statistical protocols.

For iterative and chaotic tasks, natural metrics include distances between trajectories as a function of time. One can compute normed differences between continuous state vectors, Hamming distances between discrete sequences, or divergence measures between probability distributions over outputs. These differences can be summarized across time by quantities such as average divergence, maximum divergence, or time to exceed a specified threshold. The reference trajectory provides a baseline; hardware-induced deviations are measured relative to it.

For combinatorial and reasoning tasks, metrics include success rates on instance sets, distributions of solve times, and quality measures such as solution optimality or proof length. The reference implementation defines baseline performance metrics on each task family. Hardware runs are then compared to those baselines across many instances, with careful control of randomization to avoid confounding factors.

Statistically, the key is to distinguish persistent structure from noise. This typically involves running many independent trials for each hardware–task combination, computing empirical distributions of metrics, and applying hypothesis tests or confidence interval estimates. For example, one might test whether the success rate of a particular configuration differs significantly from the reference rate at a given significance level, correcting for multiple comparisons across many devices and tasks.

Temporal stability is also important. An effect that appears only in a brief window during early evaluation may be less convincing than one that persists across time and across repeated experimental campaigns. Similarly, effects that depend on fine-grained tuning of experimental conditions must be scrutinized more carefully than those that appear robust under moderate variations in temperature, voltage, or workload.

Finally, the protocols must explicitly guard against known sources of spurious differences. This includes ensuring that identical input distributions and prompts are used across runs, that random seeds are handled consistently, and that no unintended differences in software versions or configuration options creep in between the reference and hardware runs. Only after these factors are controlled should any residual performance differences be attributed to substrate effects.

By combining sensitive metrics, robust statistical methods, and stringent control of confounding variables, the digital reference baseline becomes more than just a convenience. It becomes the anchor for a disciplined experimental program in which claims about substrate-coupled intelligence can be evaluated with the same rigor applied to other empirical questions in science and engineering.

5. Heterogeneous Hardware Execution Pool

The digital reference baseline defines what a fixed model “should” do under idealized numerical conditions. The next ingredient is a pool of real devices capable of deviating from that ideal in many different ways. The aim is not simply to have more compute, but to construct a heterogeneous substrate space in which a meta-learner can explore, and in which systematic differences can be measured and interpreted. This section describes how to assemble such a device pool, how to vary operating conditions in a controlled way, how to orchestrate experiments reproducibly, and what patterns of variation we should expect to see before invoking any notion of substrate-coupled intelligence.

5.1 Assembling a diverse device set: vendors, process nodes, and ages

A meaningful search for substrate effects requires diversity. If all accelerators are of the same model, from the same lot, run under identical conditions, the space of observable hardware behavior is narrow. By contrast, a deliberately heterogeneous pool can expose the model and the meta-learner to a wider range of physical quirks and implementation choices.

At one level, diversity means including devices from multiple vendors and product lines. Different GPU and accelerator families make different design decisions about arithmetic units, memory hierarchies, interconnects, and power delivery. Even if they implement the same nominal instruction set and floating-point formats, the microarchitectural details can differ in ways that affect timing, rounding, and noise.

Within each vendor, it is useful to include multiple process nodes and generations. Older devices may have larger geometries, different leakage characteristics, and more conservative timing margins. Newer devices may push closer to the limits of what is feasible, with tighter margins, more aggressive clocking, and narrower voltage ranges. These differences can change how non-idealities manifest.

Age and usage history also matter. Devices that have spent years in production may exhibit different thermal behavior, aging effects, or accumulated degradation compared to fresh hardware. Including both well-used and new devices allows experiments to probe whether substrate effects drift over the lifetime of a chip.

Finally, the pool can be extended to include nontraditional accelerators, such as specialized matrix-multiply ASICs, FPGAs configured for neural workloads, or prototype analog and neuromorphic devices. While the main focus may remain on mainstream digital hardware, these additional substrates expand the space in which any anomalous behavior might appear.

5.2 Controlled variation: voltage, frequency, temperature, and kernel choices

Having a diverse set of devices is only part of the story. Many important substrate effects emerge not from static properties of the chip but from how it is operated. To give those effects an opportunity to manifest, the experimental framework must support controlled variation of operating conditions.

Dynamic voltage and frequency scaling is a primary lever. Within vendor-specified safe ranges, one can vary core and memory voltages and clocks, exploring performance points from conservative, low-frequency, high-margin regimes to more aggressive, high-frequency, low-margin ones. Near the edges of specification, timing margins shrink and sensitivity to variation increases, potentially making subtle effects more visible.

Temperature is another critical parameter. By adjusting cooling solutions, airflow, and ambient temperature, or by scheduling workloads in ways that produce sustained heating or cooling, one can probe how device behavior changes with thermal state. Some substrate effects may only appear at specific temperature bands or in the presence of strong gradients across the die.

Kernel choices add a digital dimension to this variation. Many linear algebra libraries and deep learning frameworks offer multiple algorithms for the same logical operation, differing in tiling strategies, accumulation orders, or exploitation of special hardware features. By selecting among these kernels, or by introducing custom variants designed to stress particular parts of the chip, one can alter how computation traverses the physical substrate.

To maintain experimental integrity, these variations must be applied in a controlled fashion. Each run should record the exact operating point: voltage, frequency, temperature conditions, chosen kernels, and any relevant configuration flags. This makes it possible to correlate observed performance differences with specific operating conditions and to distinguish genuine substrate effects from incidental changes in software configuration.

5.3 Execution harness: reproducible job submission, logging, and versioning

The heterogeneous pool and operating knobs are only useful if experiments can be orchestrated reproducibly and observed in detail. This requires an execution harness that sits between the task suite and the hardware, managing job submission, resource allocation, and comprehensive logging.

Reproducible job submission begins with standardized experiment descriptors. Each experiment should specify the model version, task subset, input seeds, device selection policy, operating conditions, and kernel choices. The harness translates these descriptors into concrete jobs that are dispatched to the hardware pool. By storing descriptors in a version-controlled repository, researchers can rerun exact experiments or construct systematic families of experiments for comparison.

Logging is central. For each job, the harness should record not only inputs and outputs but also hardware metadata and telemetry. This includes device identifiers, firmware and driver versions, library versions, actual voltage and frequency readings, temperatures over time, power draw, and any performance counters available from the hardware. Logs should be timestamped and linked to the corresponding experiment descriptor.

Versioning applies both to software and to models. Changes in compiler versions, libraries, or operating system components can alter numerical behavior independently of hardware. The harness must track these versions alongside model weights and configuration. Only by pinning software environments can one distinguish substrate effects from software-induced differences.

The harness should also provide mechanisms for deterministic replay. If a particular run shows anomalous behavior, researchers must be able to recreate the exact conditions, including hardware selection, operating point, task instances, and random seeds. This capability is essential for moving from anecdotal observations to controlled studies, and for enabling independent teams to verify findings on their own hardware pools.

In effect, the execution harness turns a cluster of heterogeneous devices into a laboratory instrument. It abstracts the complexity of the underlying infrastructure while preserving the detail needed to interpret experimental results in terms of both digital behavior and physical context.

5.4 Baseline cross-hardware evaluation and expected patterns of variation

Before searching for exotic anomalies, it is important to establish what normal variation looks like across the hardware pool. Baseline cross-hardware evaluation provides this context. It involves running the frozen model, without any meta-controller intervention, on each device under a standard set of conditions and tasks, and comparing the results to the digital reference and to each other.

Under ordinary circumstances, we expect certain patterns. Numerical outputs will differ slightly between devices due to implementation differences in arithmetic units, variations in reduction order, and small timing-induced deviations. These differences should be small in magnitude and largely unstructured. On iterative tasks, they may cause trajectories to diverge gradually, but statistical properties of the outputs should remain similar across devices.

Performance metrics on combinatorial and reasoning tasks should also show some variation. Devices with higher throughput or lower latency may complete more instances within a fixed time budget, and minor numerical differences may tip borderline cases one way or another. However, success rates and other aggregate metrics should cluster tightly around the reference performance, with no single device or small subset consistently outperforming the others by a large margin.

Baseline evaluation can be used to construct empirical distributions of metrics for each device and for the pool as a whole. These distributions define a notion of normal hardware-induced variation. They also provide an opportunity to identify obvious outliers caused by defective hardware, misconfigured systems, or software misalignments, which can then be corrected or excluded.

Only against this backdrop does it make sense to interpret subsequent findings. If, after introducing controlled variation and meta-learning, a particular hardware–controller configuration shows performance that lies well outside the baseline distributions in a consistent and reproducible way, then it becomes a candidate for substrate-coupled intelligence or at least for interesting unmodeled device behavior. Without the baseline, such claims would be impossible to calibrate.

6. Meta-Learner as Substrate Explorer

Up to this point, the model and the hardware have been treated largely as separate entities. The model is fixed and evaluated across a diverse pool of devices, whose behavior is probed by carefully designed tasks and baselines. To move from passive observation to active exploration, the next step is to introduce a meta-learner that can deliberately choose how computation is mapped onto the substrate. Instead of treating hardware as a static background, the meta-learner treats it as a structured space of actions.

The central idea is simple. Given a fixed base model, one can still vary a great many aspects of how inference is carried out: which device is used, which kernel variants are selected, what operating point the device runs at, and potentially even how the computation is partitioned in time and space. A controller with access to these choices can explore patterns of hardware usage in search of improved task performance. If any substrate-coupled advantages exist, the controller becomes the mechanism by which they are discovered.

This section describes how to expose hardware and kernel choices as an action space, how to define training objectives for the controller, how reinforcement learning and evolutionary strategies can be used for policy search, and how to detect when a learned policy is overfitting to idiosyncrasies of a specific device rather than uncovering robust, generalizable effects.

6.1 Exposing hardware and kernel choices as an action space

To enable a meta-learner to act as a substrate explorer, hardware and implementation choices must be represented explicitly as controllable actions. This requires lifting decisions that are ordinarily made by schedulers, compilers, or configuration files into a space that the controller can manipulate at run time.

At the coarsest level, the action space includes device selection. The controller can choose among multiple accelerators of different types, generations, and usage histories. Each device can be represented by an identifier and a small set of descriptors, such as vendor, architecture family, and nominal performance characteristics. The controller's action is to assign a particular task episode to one or more devices in the pool.

Finer-grained actions involve kernel selection. Many deep learning operations can be implemented by multiple kernels that differ in tiling strategy, memory access patterns, accumulation order, or use of special instructions. The controller can be given a set of kernel choices for key operations such as matrix multiplication, convolution, and normalization. Actions then specify which kernel variant to use for each operation or for groups of operations.

An additional dimension of action is operating point configuration. Within safe limits, the controller can request particular voltage and frequency settings or ask for specific power and thermal management modes. These actions adjust how aggressively the hardware is driven, potentially exposing different timing behaviors and noise characteristics.

In more advanced designs, actions might include decisions about batch size, microbatching patterns, or even explicit insertion of small perturbations into weights or activations. These choices alter how computation is arranged in time and space on the device, providing the controller with opportunities to exploit complex interactions between workload and substrate.

To keep the problem tractable, the action space must be structured. Not all combinations of device, kernel, and operating point are valid or sensible. Constraints can be encoded to limit actions to safe and feasible configurations. The controller does not need to know the full physical meaning of its actions; it only needs consistent interfaces and feedback to learn which patterns are beneficial for the tasks at hand.

6.2 Training objectives: maximizing task performance with a fixed base model

The controller's purpose is not to modify the base model but to improve how the model performs on tasks by choosing where and how it runs. Training objectives must therefore be defined in terms of task performance metrics that can be influenced by substrate-level choices.

For each task episode, the controller selects an action that determines the hardware configuration for that episode. The base model is then executed under those conditions,

producing outputs and performance metrics. Depending on the task, these metrics might include accuracy, success on a combinatorial problem, time to solution, quality of a reasoning chain, or reward obtained in a simulated environment. The key is that these metrics reflect the end-to-end performance of the system, not low-level hardware counters.

The controller's objective is to maximize expected performance over a distribution of tasks. This can be framed as a reinforcement learning problem, where each episode consists of choosing hardware-related actions, observing the resulting task performance, and receiving a scalar reward. Alternatively, it can be treated as a black-box optimization problem over a high-dimensional action space.

Because the base model is fixed, any improvement in performance relative to the digital reference must come from two sources. One is better exploitation of known digital behavior, for example by choosing kernel implementations that reduce numerical error or reduce latency. The other is genuine substrate-coupled advantages, where physical quirks of the device under specific operating conditions provide additional structure that the controller learns to exploit.

Training objectives can be designed to highlight different aspects of performance. A simple objective might maximize accuracy or success rate. More nuanced objectives could combine correctness with time or energy cost, encouraging the controller to find configurations that are both effective and efficient. For tasks with multiple difficulty levels, the objective might emphasize success on the hardest instances, where any substrate-coupled advantage would be most visible.

Crucially, the base model is not updated during this process. Its parameters and architecture remain as defined in the reference. This isolates the effect of substrate exploration. If performance improvements are observed, they can be attributed to the controller's selection of hardware configurations rather than to changes in the model itself.

6.3 Reinforcement learning and evolutionary strategies for policy search

Searching the space of substrate actions is a challenging optimization problem. The mapping from actions to performance is noisy, delayed, and non-differentiable. Exact gradients with respect to device choices or low-level kernel configurations are not available. This makes reinforcement learning and evolutionary strategies natural candidates for training the controller.

In a reinforcement learning formulation, the controller is a policy that maps task descriptors and possibly observed hardware state to actions. Task descriptors might include information about the problem instance, desired time budget, or difficulty level. Hardware state could include recent temperatures, utilization levels, and other telemetry. The policy can be parameterized by a neural network or another flexible function approximator.

Episodes proceed as follows. A task is sampled from the distribution of interest. The policy observes the task descriptor and current hardware state and selects an action specifying device, kernels, and operating point. The base model is executed under these conditions, producing performance metrics and a scalar reward. The experience tuple is then used to update the policy via standard reinforcement learning algorithms, such as policy gradients, actor–critic methods, or more specialized bandit algorithms when the action space is relatively small.

Evolutionary strategies offer an alternative that can be simpler to implement and more robust to noise in highly irregular landscapes. In this approach, a population of candidate policies or configurations is maintained. Each candidate is evaluated on a subset of tasks, and performance metrics are used to select and mutate candidates. Over many generations, the population evolves toward configurations that perform well on average. Evolutionary methods are particularly attractive when the action space is discrete and combinatorial, or when one wants to explore widely before committing to a particular region of the space.

Both approaches must contend with the danger of overfitting to specific devices or transient conditions. If training is conducted using only a small subset of hardware or under narrow operating conditions, the resulting policy may perform well there but fail to generalize. To mitigate this, training should involve multiple devices, varied operating points, and shuffled task distributions. Regularization techniques, such as entropy bonuses in reinforcement learning or diversity-promoting mechanisms in evolutionary strategies, can also help prevent premature convergence to brittle solutions.

Ultimately, the choice between reinforcement learning and evolutionary strategies is a pragmatic one. The important point is that both treat the system as a black box whose behavior can be sampled under different actions. They provide systematic ways to search for policies that improve performance, whatever the underlying mechanisms might be.

6.4 Detecting non-portable policies: when a controller overfits to a specific device

A central concern in substrate exploration is non-portability. A policy that appears to improve performance might in fact be exploiting narrow, device-specific quirks that do not generalize to other hardware or to slightly different operating conditions. While such behavior is itself of interest as evidence of substrate coupling, it also poses risks for reliability and safety if adopted uncritically.

Detecting non-portable policies requires explicit tests. One straightforward approach is cross-device evaluation. After training a controller using a particular subset of devices, the same policy is evaluated on different devices from the pool, ideally with similar nominal capabilities but different manufacturing histories or microarchitectural details. If the policy’s performance

gains disappear or reverse on these new devices, it suggests that the controller has overfit to specific substrates.

Similarly, one can test generalization across operating points. If a policy was trained at certain voltage and frequency settings, it can be evaluated at slightly different settings within the safe range. Robust policies that exploit general features of the hardware should retain much of their advantage, whereas brittle, overfitted policies may fail when timing margins or thermal conditions change.

Another diagnostic is to compare learned policies to simpler baselines. For example, a random device selection baseline or a heuristic policy that chooses devices based on high-level metrics such as nominal performance or temperature can be evaluated alongside the learned controller. If the controller's advantage over such baselines is only visible under a narrow set of conditions, that suggests device-specific overfitting rather than broad substrate-coupled intelligence.

The execution harness described earlier plays a crucial role here. Because it records software versions, device identifiers, operating conditions, and detailed logs, it allows experiments to be repeated in controlled ways while varying only one factor at a time. This makes it possible to isolate the effect of changing the physical device from changing the policy or task distribution.

Importantly, detecting non-portable policies is not solely a matter of filtering out undesirable behavior. It is also a diagnostic tool for identifying where substrate effects are strongest. If a policy performs unusually well on a particular device and poorly elsewhere, that device becomes a candidate for deeper investigation by the anomaly and physics lab. Non-portability, in this sense, is both a warning sign and a pointer toward interesting physical behavior.

In practice, any policy intended for deployment would need to satisfy stricter criteria. It should perform well across the range of hardware and operating conditions expected in production, and it should be robust to moderate changes in the substrate. Policies that fail these tests might still be valuable for research, as windows into the interplay between computation and matter, but they would require careful containment and monitoring to avoid unintended consequences.

7. Anomaly Detection and Reduction to Physics

If the meta-learner discovers hardware–task configurations that appear to outperform the digital reference, the project leaves the realm of simple benchmarking and enters the realm of anomaly detection. At this point, the question is no longer whether some configuration does better or worse on a task, but why. Is the improvement an artifact of statistics, a previously unrecognized software issue, a subtle manifestation of known device physics, or a hint of something genuinely new?

To answer this, the framework needs a structured pathway for taking complex, model-level observations and progressively reducing them to simpler, more controlled physical experiments. This section describes that pathway. It begins with statistical methods for identifying candidate anomalies in the sea of experimental data. It then outlines how to strip away model and software complexity to isolate the underlying effect in minimal micro-benchmarks. It discusses the role of physical instrumentation in characterizing the behavior of suspect devices. Finally, it considers how to distinguish between known device physics and behavior that genuinely challenges current understanding.

7.1 Statistical tests for identifying candidate hardware-task anomalies

The first task in anomaly detection is to separate interesting deviations from the ordinary variation inherent in large-scale experiments. The heterogeneous hardware pool, diverse task suite, and meta-learning controller generate a large volume of performance data. Without careful statistical analysis, it is easy to mistake random fluctuations or selection effects for meaningful patterns.

At a high level, anomaly detection begins by defining null models of expected behavior. For each task family and metric, baseline distributions are estimated using the digital reference implementation and cross-hardware evaluations without specialized controllers. These distributions capture normal differences due to numerical variation, hardware performance, and noise. They provide reference ranges against which new configurations can be compared.

When a particular hardware–controller configuration appears to show improved performance, its metrics are evaluated relative to these baselines. Multiple episodes are run to obtain empirical distributions of performance. Standard techniques such as hypothesis testing, confidence interval estimation, and bootstrap resampling can be used to assess whether the observed differences are likely to have arisen by chance. Corrections for multiple comparisons are essential, since many hardware–task–controller combinations may be tested.

Temporal analysis also plays a role. An effect that appears only in a small window of time and then disappears may be due to transient factors such as system load or temperature fluctuations. By tracking performance over extended periods and across repeated experimental campaigns, one can distinguish stable patterns from ephemeral ones.

In addition, anomaly detection should consider correlations across tasks and devices. A configuration that improves performance on one task but degrades it on others might reflect task-specific overfitting rather than a substrate-level advantage. Conversely, a configuration that consistently improves performance across multiple related tasks on the same device is more likely to indicate a genuine interaction between model, controller, and substrate.

The output of this stage is not a final verdict but a prioritized list of candidate anomalies. These are hardware–task–controller combinations that show statistically significant, reproducible, and nontrivial deviations from expected behavior, and that therefore warrant deeper investigation.

7.2 Stripping away complexity: from full models to minimal micro-benchmarks

Complex models and tasks are valuable for discovering anomalies but not ideal for understanding them. A large neural network running a sophisticated benchmark involves many layers of abstraction, making it difficult to pinpoint which part of the system is interacting with the substrate in an unusual way. Once a candidate anomaly is identified, the next step is to simplify.

The guiding principle is reduction. Starting from the full model and task where the anomaly was observed, one systematically strips away complexity while trying to preserve the effect. For example, if a particular device and controller configuration improves performance on a family of reasoning tasks, one might first narrow the tasks to simpler instances that exhibit similar behavior. If the anomaly is associated with specific layers or operations in the model, one can construct smaller models that isolate those components.

At the extreme, the aim is to design micro-benchmarks that exercise only the suspected hardware features. These might be simple matrix multiplications with specific operand patterns, sequences of loads and stores with controlled timing, or small neural network layers with fixed weights. The idea is to produce a tight loop of operations that can be run many times, under controlled conditions, to explore the behavior of the device in a clean, focused way.

Throughout this reduction process, one continually checks whether the anomaly persists. If it disappears when the model is simplified or the task is narrowed, that suggests that the original effect depended on higher-level interactions and may not be easily reducible to basic physical mechanisms. If it survives, then the simplified benchmark becomes the new focus of analysis.

This reduction is not purely mechanical. It requires hypotheses about which aspects of the computation are most likely to interact with the substrate: particular arithmetic patterns, memory access structures, or timing relationships. Those hypotheses can be informed by hardware knowledge, but they can also be explored systematically by generating families of micro-benchmarks that vary in controlled ways and observing where the anomaly is strongest.

7.3 Physical instrumentation: power traces, timing jitter, thermal profiles, and EM emissions

Once a simplified benchmark that preserves the anomaly has been developed, attention shifts to the physical side of the system. At this stage, the goal is to observe the device not just through logical outputs, but through its physical behavior: how it draws power, dissipates heat, responds over time, and interacts with its electromagnetic environment.

Power traces are a natural starting point. By measuring instantaneous power consumption at high temporal resolution while the benchmark runs, one can detect patterns such as periodic fluctuations, spikes, or correlations with input patterns. Differences between power traces for reference and anomalous configurations may indicate that the device is operating in different regimes or that certain resources are being stressed in unusual ways.

Timing measurements provide another window. Using high-resolution timers, one can measure the latency of individual operations or sequences under different conditions. Jitter and variability in these timings can reveal marginal paths, metastable behavior, or interactions between computation and power delivery. Comparing timing profiles across devices and operating points can help locate where in the execution pipeline anomalies arise.

Thermal profiling adds a spatial dimension. Sensors embedded in the device, external infrared cameras, or on-board thermal diodes can be used to track temperature distributions over time. If the anomalous behavior is associated with particular regions of the die heating or cooling in distinct ways, that suggests localized activity patterns or hot spots that may be linked to the observed effects.

Electromagnetic emissions, while more challenging to measure, can provide further information. Probes placed near the device can detect radiated or conducted signals that correlate with specific computational patterns. Differences in emission spectra between normal and anomalous runs may indicate that certain circuits or interconnects are oscillating or coupling in unexpected ways.

Collecting these physical measurements requires careful experimental design. Instrumentation can itself perturb the system, and measurements must be synchronized with the logical execution of the benchmark. Nonetheless, the combination of logical and physical observations is essential for moving from the abstract notion of a substrate effect to concrete hypotheses about which physical mechanisms might be involved.

7.4 Disentangling known device physics from genuinely unexplained behavior

With micro-benchmarks and physical measurements in hand, the final step is interpretation. The question becomes whether the observed behavior can be explained using existing models of device physics and circuit behavior, or whether it points to something genuinely beyond current understanding.

The first line of analysis is to consult and refine existing models. Semiconductor device equations, power distribution models, timing analysis, and electromagnetic simulations can all be brought to bear. If, for instance, an anomaly appears to be associated with voltage droop under high load, known models of power delivery networks and dynamic IR drop may be sufficient to explain the behavior. If certain arithmetic patterns produce biased results on one

device but not others, analysis of the implementation of floating-point units may reveal undocumented but conventional design choices.

Often, anomalies will turn out to be previously unmodeled but conceptually ordinary effects: interplay between process variation and marginal timing, subtle interactions between clock domains, or corner cases in numerical libraries. In such cases, the appropriate response is to refine models, update design practices, and possibly adjust operating guidelines. The anomaly then joins the catalog of known behaviors.

More interesting are cases where, after exhaustive application of known models and simulations, a residual effect remains unexplained. For an effect to be considered genuinely anomalous in a deeper sense, several conditions should be met. The behavior must be robust across repetitions and independent measurements. It must persist under variations in tasks and micro-benchmark designs that preserve the suspected underlying mechanism. It should be observed on multiple devices of similar design, ideally from different lots and in different environments, to rule out idiosyncratic defects or local interference.

Even then, caution is warranted. Modern hardware and software stacks are extraordinarily complex, and subtle interactions can masquerade as fundamental anomalies. Only when all plausible conventional explanations have been systematically tested and ruled out does it become reasonable to suggest that the behavior challenges existing physical models.

From the perspective of this framework, however, it is not necessary to resolve such questions definitively. The primary goal is to provide a disciplined pathway for moving from high-level AI performance anomalies to concrete physical hypotheses and experiments. Whether the outcome is an expanded understanding of known device physics or a rare hint of something deeper, the method ensures that claims about substrate-coupled intelligence are grounded in empirical evidence and careful reasoning rather than conjecture.

8. From Defects to Design: Co-Design Implications

If substrate-coupled effects turn out to be real in practice, they should not be viewed only as anomalies to be suppressed. They are signals about how computation and matter interact under real workloads. Some of those interactions will be undesirable and must be engineered away. Others will be neutral background. A small subset may be worth cultivating deliberately. This section explores how to move from a reactive stance, where defects are merely detected and corrected, to a proactive stance, where the lessons of substrate coupling inform the co-design of future algorithms and hardware.

The central shift is conceptual. Instead of assuming that the goal is a perfect emulation of an abstract numerical model, we accept that any physical implementation will have its own character. The task then becomes twofold: to ensure that this character does not undermine reliability and safety, and to identify ways in which it might be harnessed constructively, whether by changing hardware, changing algorithms, or both.

8.1 Classifying substrate coupling: harmful, benign, and potentially useful

Once substrate-level effects are detected and characterized, they need to be classified. Not all coupling between model and device warrants the same response. Some forms of coupling are unambiguously harmful, introducing errors or instabilities that threaten correctness or safety. Others are effectively benign, adding small perturbations that neither help nor hurt in any material way. A third class may be potentially useful, providing structure or diversity that can, under controlled conditions, enhance performance.

Harmful coupling includes effects that produce silent data corruption, unpredictable behavior under plausible operating conditions, or strong non-portability across devices that are intended to be interchangeable. Examples might include timing-induced logic errors that only appear under certain workloads, power-related glitches that flip bits in critical paths, or device-specific numerical biases that cause systematic misclassification in safety-critical systems. For these effects, the design goal is straightforward: detect them early, understand their causes, and either redesign hardware and software to eliminate them or constrain operating regimes so they cannot occur.

Benign coupling encompasses effects that are present but effectively drown in the noise of other sources of variation. Slight differences in rounding behavior across devices, small temperature-dependent changes in latency that do not significantly affect model outputs, or minor variation in power consumption patterns may all fall into this category. These effects should be documented and monitored, but they do not justify substantial architectural changes unless they are found to correlate with more serious problems.

Potentially useful coupling is the most subtle category. Here, the substrate contributes something that can legitimately improve performance on some tasks, without introducing unacceptable risks. This might take the form of structured noise that enhances generalization, device-specific variation that enriches an ensemble of models, or analog-like transformations that act as implicit feature maps. The defining feature is that the effect is both reproducible and exploitable under controlled conditions. For such effects, the design question is not how to eliminate them but how to integrate them systematically into the overall architecture and learning process.

This classification is not static. As understanding deepens, effects may move from one category to another. What initially appears harmful may, after further study, be reinterpreted as benign or even useful in constrained contexts. Conversely, what seems benign may be revealed as risky when deployed at scale. The framework of detection, reduction, and physical analysis described earlier provides the empirical basis for making and revising these judgments.

8.2 Learning with variation and noise as a first-class design principle

If some forms of substrate coupling are classified as potentially useful, the next question is how to design systems that intentionally learn with variation and noise rather than in spite of them. This idea has precedents in machine learning. Techniques such as dropout, stochastic gradient descent, and data augmentation deliberately inject randomness to improve generalization. In hardware-aware learning, process variation and analog noise can play analogous roles.

One design principle is to treat each physical device as a slightly different instantiation of a common function class. Instead of enforcing strict uniformity across devices, the training process can embrace diversity. Models or controllers can be trained to perform well on distributions over devices and operating conditions rather than on a single canonical environment. This makes the system robust to variation and creates an opportunity to exploit differences. For example, an ensemble may be constructed where each member runs on a different device, and diversity in substrate behavior contributes to the ensemble's collective performance.

Another principle is to use noise as a regularizer. If certain hardware configurations introduce small, structured perturbations into activations or gradients, training can be adjusted to expect and accommodate these perturbations. Rather than treating them as nuisances, they become part of the stochasticity that shapes the loss landscape. In some cases, intentionally operating devices in regimes where noise is slightly higher than usual may provide beneficial exploration, as long as safety bounds are respected.

To make variation and noise first-class citizens, they must be measured and modeled. Instead of relying on abstract assumptions about Gaussian noise, the system can learn empirical distributions of substrate perturbations under different operating conditions. These distributions can then be incorporated into training, either by simulating them in software or by training directly on hardware under those conditions. In this way, the physical substrate becomes part of the data-generating process that shapes the learned function.

The challenge is to balance this openness to variation with the need for predictability. It is one thing to let noise guide learning during training; it is another to rely on uncontrolled variation at deployment. A practical approach is to exploit variation during training, while constraining deployment to narrower, well-characterized operating regimes. This mirrors practices in other

domains, where systems are trained under rich conditions but certified for use under specific, documented ones.

8.3 Reproducing anomalous performance digitally: extracting algorithmic insights

Even when substrate-coupled behavior appears to improve performance, reliance on particular physical quirks may not be viable in the long run. Hardware changes, devices age, manufacturing processes evolve. A key question is therefore whether anomalous performance attributed to substrate effects can be reproduced digitally, without depending on specific devices. If it can, then the anomaly has served as a pointer to a better algorithmic strategy rather than as an enduring physical advantage.

The process begins by characterizing the conditions under which the anomalous performance appears. Does it depend on specific input structures, particular parts of the model, or narrow ranges of operating conditions? By comparing behavior with and without the anomaly, one can attempt to infer what additional transformation the substrated-coupled system is effectively performing. This might manifest as a particular kind of noise injection, a bias in numerical rounding, or a tendency to emphasize certain frequency components in signals.

Once a plausible hypothesis is formulated, it can be tested in pure software. For example, if a certain device appears to add beneficial noise to intermediate activations at specific magnitudes, the same noise can be injected deliberately in the digital reference implementation. If this reproduces the performance improvement across tasks and across hardware, then the essential ingredient has been captured digitally. Similarly, if an analysis suggests that certain kernel choices effectively reorder operations in a way that improves numerical stability, that reordering can be codified as a new software-level algorithm.

In some cases, reproducing anomalous behavior digitally may require changes to the training process rather than to inference alone. If the model has adapted to substrate-specific perturbations over time, then retraining with analogous perturbations in a hardware-agnostic environment may yield models that perform similarly without relying on those perturbations at run time.

This extraction of algorithmic insights serves multiple purposes. It reduces dependence on particular substrates, making systems more portable and predictable. It also accelerates the diffusion of beneficial behaviors. Once a useful pattern is identified and understood, it can be incorporated into standard libraries and frameworks, benefiting a wide range of applications. Finally, it sharpens the distinction between what is truly physical and what is essentially algorithmic, preventing the inflation of engineering tricks into claims about new physics.

8.4 Pathways from accidental quirks to intentional substrate-aware architectures

If the study of substrate-coupled behavior reveals that certain kinds of physical interaction are both useful and irreducible to simple digital abstractions, a longer-term path opens up: the design of intentional substrate-aware architectures. In such architectures, hardware and algorithms are co-designed so that the physical dynamics of the substrate are not merely tolerated but actively harnessed.

One pathway begins with accidental quirks discovered in existing hardware. Suppose, for example, that a particular combination of interconnect wiring and power delivery produces a subtle low-pass filtering effect that helps certain models suppress high-frequency noise in their activations. Rather than relying on this effect as an undocumented feature, future architectures could incorporate explicit analog filtering circuits or configurable signal-conditioning blocks that provide similar behavior under controlled conditions. Software interfaces would expose these features deliberately, allowing models to opt in and out.

Another pathway is to generalize from individual anomalies to broader design patterns. If studies reveal that models benefit from having access to physically diverse computational elements—some noisy, some precise, some fast, some slow—then architectures can be built with heterogeneous cores that are explicitly labeled and managed as such. Meta-learners can then be trained to assign different parts of a computation to different cores based on their characteristics, turning what was once uncontrolled variation into a designed resource.

Intentional substrate-aware architectures also invite new forms of abstraction. Rather than viewing hardware solely through the lens of instruction sets and memory hierarchies, one can define higher-level “physical primitives” such as noise sources, delay lines, oscillators, or coupled arrays, and provide programming models that let algorithms compose these primitives. This blurs the line between digital and analog computation, but in a controlled, documented way.

Throughout this evolution, safety and governance considerations remain paramount. Intentional exploitation of physical dynamics must come with rigorous characterization, clear specifications, and mechanisms for monitoring and control. Certification regimes may need to expand to include not just logical correctness but adherence to specified physical behaviors. Documentation must be clear about which aspects of computation depend on substrate properties and which are portable across devices.

In summary, the journey from defects to design passes through several stages. Initially, substrate effects are treated as unwanted anomalies to be detected and neutralized. As understanding grows, some effects are reclassified as benign background, others as potential resources. From there, the field faces a choice. It can either clamp down on all variation in the

name of uniformity or allow carefully selected forms of substrate-aware behavior to inform the co-design of future systems. The framework outlined in this paper is intended to support that choice by providing the tools needed to see, measure, and reason about how intelligence and matter meet in practice.

9. Safety, Governance, and Epistemic Discipline

Any framework that deliberately invites AI systems to explore the boundaries of their hardware substrate raises a distinct set of safety and governance questions. Even if the primary motivation is scientific—to understand whether real devices can contribute anything beyond the digital model—the process can uncover behaviors that are non-portable, fragile, or poorly understood. Without strong safeguards, such behaviors could leak into production systems, distort evaluations, or tempt practitioners into overstating what has actually been discovered.

This section addresses the institutional and methodological disciplines needed to keep substrate exploration aligned with broader goals of safety and reliability. It emphasizes sandboxing and containment for experiments, requirements for transparency and independent replication, cultural norms that resist techno-mysticism, and policy implications for certifying AI systems across heterogeneous hardware. The aim is to ensure that curiosity about substrate effects does not erode basic guarantees about how AI systems behave in the world.

9.1 Sandboxing and containment for substrate-exploring AI experiments

The first safety principle is containment. Substrate-exploring experiments should not be conducted on the same infrastructure or with the same operational assumptions as production AI services. Instead, they should be confined to dedicated experimental environments, with explicit boundaries and controls.

A sandboxed environment for substrate exploration has several characteristics. It runs on a separate cluster or on clearly segregated partitions of a cluster, with access controls that limit who can launch experiments and who can modify configurations. Experimentation tools, including the meta-learner and execution harness, are deployed here, not in the general-purpose serving infrastructure. Network access from the sandbox to external systems is restricted to what is strictly necessary for logging and monitoring; it is not used to serve user-facing queries.

Within the sandbox, resource allocation is tightly managed. Experiments that push devices toward the edges of operating envelopes—higher temperatures, more aggressive voltages, or unusual workloads—are subject to additional safety checks. Guardrails prevent the controller

from issuing actions that would drive hardware outside specified safe bounds, and hardware-level protections such as thermal throttling and overcurrent shutdown remain active.

Equally important is the separation of artifacts. Models, controllers, and configurations that have been trained or tuned in substrate-exploring contexts should be clearly labeled and not deployed into production without deliberate review. Even if a policy or configuration appears to improve performance on benchmark tasks, its dependence on uncharacterized substrate behavior makes it unsuitable for general deployment until it has been thoroughly understood and, ideally, reproduced in a substrate-agnostic way.

This containment does not exist to suppress research. It exists to ensure that experimental systems, which by design push against the boundaries of current understanding, do not leak into contexts where predictability and accountability are paramount. It mirrors practices in other high-stakes domains, where experimental reactors, prototypes, or testbeds are kept distinct from operational infrastructure.

9.2 Transparency, documentation, and the necessity of independent replication

Substrate-coupled intelligence, if it exists at all, will be subtle and easy to misinterpret. This makes transparency and documentation essential. Claims about anomalous behavior must be backed by detailed records of what was done, on which devices, under which conditions, and using which software stacks. Without such records, other researchers cannot reproduce or evaluate the findings, and governance bodies cannot assess their implications.

At the level of individual experiments, transparency means preserving experiment descriptors, logs, and code in version-controlled repositories. Each reported anomaly should be traceable back to specific experiment configurations, with enough detail for others to recreate the setup as closely as their hardware allows. This includes not only high-level descriptions of models and tasks, but also low-level details such as driver versions, compiler flags, kernel choices, and device identifiers.

Documentation should also include negative results. Many exploratory experiments will find nothing beyond ordinary variation. Recording and, where appropriate, publishing these null results helps calibrate expectations and prevents a bias toward only reporting surprising behavior. Over time, a corpus of experimental outcomes can provide a more complete picture of how often substrate effects matter in practice.

Independent replication is a core safeguard against error and overinterpretation. If an anomaly is observed in one lab on one hardware pool, the next step is to attempt replication elsewhere, using different devices, software stacks, and measurement tools. Only when multiple independent teams observe similar behavior, and agree on the conditions under which it appears, can the anomaly be treated as a robust empirical fact rather than a local curiosity.

From a governance standpoint, transparency and replication provide the basis for informed oversight. Regulators, standards bodies, and institutional review boards cannot meaningfully evaluate substrate-related risks or opportunities on the basis of proprietary claims or unpublished data. Clear documentation and accessible replication paths allow external stakeholders to scrutinize methods and conclusions, and to decide how, if at all, substrate phenomena should influence standards for AI deployment.

9.3 Avoiding techno-mysticism: treating anomalies as physics problems, not miracles

When complex AI systems behave in unexpected ways, there is a temptation to reach for mystical language. Anomalies can be described as if they revealed an emergent intelligence in the hardware, or as if the system were tapping into unknown realms of computation. Such narratives are psychologically compelling, but they undermine epistemic discipline.

In the context of substrate exploration, the guiding norm should be to treat anomalies as physics problems. This means assuming, by default, that surprising behavior arises from some combination of known but unmodeled device physics, software interactions, and statistical fluctuation. The task is to identify which of these factors is responsible, and in what proportions, rather than to leap to conclusions about new laws of nature or qualitatively new forms of intelligence.

Practically, this norm suggests several behaviors. Researchers should avoid anthropomorphic language when describing substrate effects. Instead of saying that a device “wants” to compute a certain answer or that a model “learned to use quantum tunneling,” one should describe observed correlations, measurement results, and hypotheses in neutral terms. Claims about unknown physics should only be made after thorough attempts to fit observations within existing theories have failed, and even then, they should be phrased cautiously.

The norm also implies humility about what substrate exploration can deliver. Even if substrate-coupled advantages are found on specific tasks, they do not imply that existing hardware is secretly far more powerful than current models suggest. Small performance boosts on carefully chosen benchmarks are not equivalent to breaking widely believed complexity-theoretic barriers. Overstating results can mislead both technical and non-technical audiences, and can distort funding and policy decisions.

Avoiding techno-mysticism is not about denying the possibility of genuine surprise. It is about ensuring that surprise is handled with the same rigor applied in other empirical sciences. The most interesting outcomes of substrate exploration—whether new device phenomena, improved algorithms, or revised safety guidelines—will be those that survive skeptical scrutiny, not those that generate the most dramatic headlines.

9.4 Policy implications for certifying AI systems across heterogeneous hardware

Even if substrate exploration remains largely within research settings, its findings have implications for how AI systems are certified and regulated, especially when those systems are deployed across heterogeneous hardware. If real differences in behavior can arise from substrate-level effects, then policies that treat hardware as a neutral background may need revision.

One implication is that certification regimes may need to move from certifying models in isolation to certifying model–hardware combinations, or at least ranges of hardware configurations. Today, many evaluations implicitly assume that a model's behavior is fully specified by its weights and software. If substrate effects are non-negligible, then certifications that ignore hardware could be misleading. Regulators may ask for evidence that a model performs safely and reliably across the range of devices and operating conditions in which it is expected to run, or they may restrict certified operation to specific, documented hardware environments.

Another implication concerns change management. Hardware evolves continuously. New accelerator generations, firmware updates, and process shrinks can all alter the substrate. If a system's certification depends on substrate behavior, then changes to hardware may need to trigger re-evaluation, much as significant software updates do today. This poses logistical challenges, but it also encourages closer coordination between hardware vendors, AI developers, and regulators.

Transparency about hardware becomes more important in this context. Users and oversight bodies may require clear labeling of which hardware was used in training, evaluation, and deployment, and what is known about substrate-related behavior. Standards may emerge for characterizing and reporting device-level variation, much as there are standards for reporting software dependencies.

Finally, policy may need to address intentional substrate-aware architectures. If future systems are designed to rely on specific physical dynamics for their operation, regulators will have to decide how to evaluate such systems. They may require additional testing, physical characterization, or safety margins to ensure that designed substrate behavior remains within acceptable bounds. Conversely, policies might restrict or discourage reliance on substrate phenomena that are difficult to monitor or control, favoring designs that remain as close as possible to the digital abstraction.

In all cases, the key is alignment between technical understanding and policy assumptions. If technical work reveals that substrate effects are negligible for most practical purposes, policies can safely treat hardware as interchangeable within broad classes. If, instead, substrate effects

prove significant in some domains, policy will need to reflect that reality. The framework described in this paper is intended to help clarify which of these worlds we inhabit, and to provide the empirical basis for whatever governance structures future AI systems will require.

10. Illustrative Case Studies and Thought Experiments

The preceding sections have laid out an abstract framework for detecting substrate-coupled intelligence and for reducing any observed anomalies to questions of device physics. To make the discussion more concrete, it is helpful to consider a few illustrative scenarios. These are not predictions but structured thought experiments: sketches of how the framework might behave when confronted with particular kinds of tasks and outcomes.

The first scenario imagines an “Euler-identity oracle,” where a hardware–controller configuration appears suspiciously good at discriminating between mathematically structured and unstructured expressions. The second focuses on satisfiability and related combinatorial problems, which sit near long-standing frontiers in complexity theory. The third extracts lessons for future quantum, analog, and neuromorphic platforms, which intentionally blur the line between digital abstraction and physical dynamics. The fourth outlines what a genuine discovery of unknown physics would look like in practice, and how high the evidential bar must be.

10.1 A hypothetical Euler-identity oracle: detecting task-specific anomalous gains

Consider a task built around a simple dichotomy: discriminating between mathematically true and false expressions drawn from a specific domain. For concreteness, suppose the domain includes variants of Euler’s identity and similar equalities, such as those involving trigonometric functions and complex exponentials. The task is to label each expression as true or false, given a textual or symbolic encoding.

In the reference setup, a fixed model is trained or fine-tuned to perform this classification using standard training data and high-precision arithmetic. Its performance is measured on a held-out set of expressions, including near-miss variants where terms are permuted, signs flipped, or constants perturbed. The reference model achieves a certain accuracy, reflecting its learned ability to parse and reason about such expressions.

Now place the same model into the heterogeneous hardware pool and introduce the meta-learner. The controller is allowed to choose devices, kernel variants, and operating points, but not to modify the base model’s weights. The training objective is to maximize classification accuracy on new, unseen expressions drawn from the same distribution. Over many episodes,

the controller explores different substrate configurations, gradually learning which ones tend to yield better performance.

Suppose, for the sake of the thought experiment, that a particular device–kernel–operating-point configuration begins to outperform the reference by a statistically significant margin. It is consistently more accurate on the most challenging instances, such as expressions designed to be just barely distinguishable or that require deeper chains of reasoning. The advantage persists across multiple runs and across fresh test sets. Other devices and configurations do not show this effect.

From the perspective of the framework, this hypothetical "Euler-identity oracle" is a candidate anomaly. The process in earlier sections would now unfold. First, statistical checks would confirm that the advantage is real and not a sampling artifact. Next, the anomaly would be simplified: perhaps by restricting to a narrower class of expressions, isolating which parts of the computation differ numerically, and constructing micro-benchmarks that mimic those parts. Physical instrumentation might reveal that the anomalous configuration induces consistent numeric biases in certain operations that, by accident, align with the structure of the task.

Once such a bias is understood, it may prove possible to reproduce the performance digitally—for example, by introducing an analogous bias or noise distribution in the reference implementation. If so, the "oracle" is demystified: it was not computing anything inaccessible to the digital model but was implementing a slightly different function in a way that happened to be advantageous for the task. If, after exhaustive reduction and modeling, the advantage resisted such explanations, the case would advance to the higher evidential bar described in the final subsection.

In any case, the thought experiment illustrates how very specific, task-focused gains could be detected, traced back to substrate behavior, and either absorbed into better algorithms or used as a prompt for deeper physical investigation.

10.2 Substrate-sensitive performance on SAT and other combinatorial problems

Satisfiability and related combinatorial problems provide a complementary testbed. They are well-defined, admit clear correctness criteria, and include families of instances believed to be hard for generic algorithms. While there is no reason to expect current hardware to secretly solve such problems efficiently in general, even modest, structured advantages on certain instance classes would be interesting.

In the framework, a frozen model is tasked with predicting satisfying assignments, refutations, or heuristic signals such as "likely satisfiable" versus "likely unsatisfiable" for randomly generated or carefully constructed SAT instances. A high-precision reference implementation establishes baseline performance across instance sizes and hardness regimes. The

heterogeneous hardware pool and meta-learner are then brought into play, with the controller allowed to choose substrate configurations for each instance.

Several patterns could, in principle, emerge. Perhaps certain devices and operating points yield marginally better success rates on near-threshold instances, where formula parameters are tuned to produce hard cases. Or certain kernel variants might consistently lead to shorter search trajectories in models that implement iterative refinement or guidance for a separate solver. If such patterns are stable and localized to specific substrates, they are candidate substrate-coupled effects.

The subsequent analysis would mirror that for the Euler case: simplify the models, isolate critical operations, and reduce to micro-benchmarks that exercise the relevant arithmetic and memory patterns. Physical measurements might reveal that particular configurations modulate noise, timing margins, or internal resource contention in ways that act as a kind of stochastic search enhancement. From there, one could attempt to replicate the beneficial effect in pure software, for example by adding specific noise processes or by altering reduction orders.

The key insight is that even on classic problems like SAT, the framework treats any observed advantage not as evidence of a broken complexity barrier, but as a prompt to identify what additional structure the substrate is providing. If that structure can be understood and abstracted, it becomes part of the algorithmic toolkit rather than a mysterious property of one particular chip.

10.3 Lessons for future quantum, analog, and neuromorphic AI platforms

While the discussion so far has focused on mainstream digital hardware, the same principles apply, perhaps even more strongly, to quantum, analog, and neuromorphic platforms. In those domains, the abstract algorithm and the physical substrate are intertwined by design. The architecture is built to exploit interference, continuous dynamics, or device variability, rather than to suppress them.

For quantum systems, substrate exploration is built in: the entire promise of quantum computing hinges on harnessing specific physical phenomena. Yet even here, there is a need to distinguish between the intended algorithmic effects and unmodeled substrate behavior. Decoherence, crosstalk, and calibration errors can produce apparent speedups or slowdowns that are unrelated to algorithmic complexity. Applying a framework similar to the one described in this paper means treating quantum devices as physical systems whose behavior must be carefully characterized, and whose deviations from idealized models must be reduced to concrete mechanisms rather than ascribed to “quantum magic.”

Analog accelerators and neuromorphic chips occupy a middle ground. They intentionally exploit device physics for computation but still present high-level programming models. For such

systems, substrate exploration is not an optional, speculative activity but a core part of design and validation. The techniques outlined—building high-precision references, designing sensitive task suites, employing meta-learners to explore operating regimes, and reducing anomalies to micro-benchmarks and physical models—can be adapted directly. They help ensure that the behavior of such systems is understood well enough to be used safely and reproducibly, even as they leverage physical dynamics.

The broader lesson is that as AI hardware becomes more varied and more tightly coupled to its physics, the need for rigorous, physics-aware evaluation and governance increases. The discipline of distinguishing between algorithmic advantages and substrate-specific quirks will be as important for quantum and neuromorphic systems as it is for today's digital accelerators, perhaps more so.

10.4 What a genuine discovery of unknown physics would look like in practice

Finally, consider the most dramatic possibility: that substrate exploration uncovers a phenomenon that cannot be reconciled with existing physical theories. What would such a discovery look like in practice, and how would it differ from the more modest anomalies and effects discussed so far?

First, a genuine discovery would emerge only after the full sequence of reduction and analysis. An initial performance anomaly would be identified and statistically validated. Complexity would be stripped away until a minimal micro-benchmark preserving the effect was found. Physical instrumentation would document how the device behaves during the effect, across power, timing, thermal, and electromagnetic domains.

Second, all plausible conventional explanations would be systematically explored and ruled out. This includes process variation, marginal timing, undocumented microarchitectural behavior, software bugs, data leakage, misconfigured randomness, and measurement errors. Known device physics would be brought to bear through simulation and modeling. Only when these avenues failed to account for the observations would the phenomenon be considered unexplained in a deeper sense.

Third, independent replication would be essential. Multiple teams, using different hardware pools, instruments, and software stacks, would need to observe the same effect under similar conditions. Replication would include controlled changes to device design, operating points, and environmental conditions to probe where the effect appears and disappears. The phenomenon would need to be robust to such variations, within a well-characterized regime.

Fourth, the effect would need to be formulated precisely enough to pose a challenge to existing theory. It might manifest as a reproducible violation of expected noise statistics, as an apparent capacity to transmit information or perform computation beyond known bounds, or as an

energy–time tradeoff that contradicts established limits. In each case, the claim would be articulated in terms familiar to physicists, not just to AI practitioners.

Finally, the community's response would resemble the response to any major empirical challenge in physics: cautious, incremental, and debate-filled. New theoretical proposals would attempt to explain the effect. Further experiments would be designed to test competing hypotheses. Over time, either a revised understanding would emerge that incorporates the phenomenon into an expanded theory, or the effect would be traced to some previously overlooked conventional mechanism.

From the vantage point of this framework, the point is not to predict that such discoveries will occur, nor to hinge the value of substrate exploration on their likelihood. Rather, it is to ensure that if something truly surprising does happen, it is handled with the right kind of discipline. A genuine discovery of unknown physics would be the end of a long process of careful elimination and replication, not the starting assumption. The same tools that help distinguish harmful from useful substrate coupling, and algorithmic from physical advantages, also provide the scaffolding for recognizing and integrating truly new physical phenomena, should they ever appear in the computational substrates that underlie our AI systems.

11. Conclusion

The question that motivated this paper was deliberately narrow and concrete: if there were any way for an AI system to lean on the physical quirks of its hardware to improve performance, how would we detect it, characterize it, and distinguish it from ordinary implementation noise? Rather than assuming that such substrate-coupled intelligence exists, we have focused on building the conceptual and experimental scaffolding that would make it empirically meaningful to look for it—and to understand what it would mean if we did or did not find it.

The result is a framework that treats model and device as a joint physical system, anchored by a high-precision digital reference, exercised by a diverse task suite, probed by a heterogeneous hardware pool, and explored by a meta-learner whose actions are confined to substrate-level choices. Around this core, we have wrapped an anomaly pipeline that reduces model-level surprises to micro-benchmarks and, where necessary, to questions of device physics, and a governance layer that emphasizes containment, transparency, and epistemic discipline.

11.1 Summary of the proposed framework and its intended contributions

The framework begins with a deterministic, high-precision implementation of a fixed model. This reference serves as the digital ground truth: what the model does when the underlying arithmetic is as clean and well-controlled as we can practically make it. On top of this reference,

we build a task suite that includes both iterative or chaotic rollouts, which amplify tiny numerical differences into observable divergences, and structured combinatorial and reasoning benchmarks, which directly probe capability on nontrivial problems.

The next layer is a heterogeneous hardware execution pool. By assembling devices from different vendors, process nodes, and ages, and by controlling operating conditions such as voltage, frequency, temperature, and kernel choices, we create a rich substrate space in which potential effects can manifest. An execution harness standardizes how experiments are launched, logged, and versioned, so that device-specific behaviors can be tied back to precise configurations.

A meta-learner then acts as a substrate explorer. With the base model frozen, the controller's action space consists of hardware- and implementation-level choices: which device to use, which kernels to invoke, and how aggressively to drive the hardware. Its training objective is to maximize task performance under these constraints. If any substrate-coupled advantages exist, the controller is the mechanism that might discover them.

Finally, an anomaly pipeline and physics lab close the loop. Candidate anomalies are identified via statistical tests against baseline cross-hardware behavior. Complexity is stripped away until minimal micro-benchmarks that preserve the effect are obtained. Physical instrumentation—power, timing, thermal, and electromagnetic measurements—is used to connect logical anomalies to concrete device behavior. Known device physics is brought to bear to explain as much as possible; only residual effects that withstand this scrutiny are treated as genuinely puzzling.

The intended contributions of this framework are threefold. First, it provides a disciplined way to test for substrate-coupled intelligence without presupposing its existence. Second, it offers a route to better reliability and portability, by revealing where model behavior depends on unmodeled device details. Third, it supplies a bridge between AI and device physics, enabling constructive feedback between anomalies discovered in AI workloads and improvements in hardware design and modeling.

11.2 Limits of detection: what this system cannot rule out

No experimental framework is omniscient. It is important to be explicit about what this system cannot detect or rule out, even if implemented at scale.

First, the framework is limited to those substrate effects that can be excited and observed under the chosen tasks, hardware pool, and operating regimes. If there are physical phenomena that only appear at voltages, temperatures, or timescales outside safe operating conditions, they will remain invisible. Likewise, if certain effects require very specific input structures or workload patterns that are not present in the task suite, they may never be triggered.

Second, detection is constrained by measurement resolution. The high-precision reference reduces numerical noise but does not eliminate it entirely. Hardware telemetry has finite granularity and bandwidth. Statistical tests have finite power. Weak substrate effects that produce only tiny, sporadic performance shifts may be indistinguishable from background variation, especially when aggregated across many tasks and devices.

Third, the interpretive pipeline assumes that the dominant physics of the devices is well described by existing theories, and that anomalies can be localized by progressive reduction. If reality includes higher-order interactions or system-level couplings that are only visible in full, complex workloads, some behaviors might resist clean reduction to micro-benchmarks without ceasing to exist.

Fourth, the framework does not resolve deep questions in complexity theory. Even if no substrate-coupled advantages are found on combinatorial tasks like SAT within the tested regimes, that does not prove that no physical substrate could ever provide such advantages. It merely indicates that, within the tested hardware and conditions, nothing beyond ordinary algorithmic behavior was detected.

Finally, the system cannot prevent misinterpretation on its own. Even with careful baselines and logging, human actors can overstate weak or ambiguous findings, or under-report null results. The framework lowers the barrier to disciplined conclusions, but it does not guarantee that all conclusions drawn from it will be disciplined.

11.3 Future research directions in substrate-aware AI and experimental design

Several lines of future work suggest themselves.

One direction is to expand and diversify the hardware and task spaces. Including emerging accelerators, such as in-memory compute devices, more aggressive analog units, or early neuromorphic chips, would test the framework under conditions where substrate dynamics are deliberately more prominent. On the task side, incorporating richer control problems, multi-agent simulations, or long-horizon reasoning tasks could reveal subtler interactions between computation, feedback, and device behavior.

Another direction is to refine the meta-learner itself. More sophisticated controllers that can adapt over longer timescales, incorporate richer hardware telemetry, or coordinate multiple devices concurrently may be better positioned to discover substrate-coupled patterns, if they exist. At the same time, research is needed on regularization and generalization strategies that discourage brittle, non-portable policies and emphasize robust patterns that transfer across hardware.

A third line of work concerns the anomaly pipeline. Techniques from causal inference, experimental design, and scientific machine learning could be brought to bear on the reduction process, helping to systematically generate and prioritize micro-benchmarks that are most informative for distinguishing among competing hypotheses. Better integration between AI evaluation and EDA (electronic design automation) tools could also help translate anomalous behaviors directly into updated device models or design constraints.

On the governance side, there is room to develop standard protocols and benchmarks for substrate-aware evaluation. Just as the community has converged on suites of tasks for measuring model-level capabilities, it may eventually converge on standardized procedures for characterizing hardware–model interactions, complete with reporting formats that capture both digital and physical aspects of system behavior.

Finally, there is conceptual work to be done connecting this framework to broader theories of computation and intelligence. Even if all observed substrate effects prove explainable in conventional terms, the exercise of seeing intelligence as a property of coupled algorithm–matter systems may influence how we think about co-design, robustness, and the limits of abstraction in AI engineering.

11.4 Closing reflections on intelligence, matter, and the discipline of being surprised

At a high level, the motivation for this work is not to conjure mysterious new sources of intelligence out of silicon, but to respect the fact that all computation is physical. Models do not run on Turing tapes; they run on devices with histories, imperfections, and dynamics. Ignoring that reality is an approximation that has served computer science well, but it is still an approximation. As AI systems grow more capable and more entangled with the world, it becomes increasingly important to understand where that approximation holds and where it starts to fray.

The framework outlined here advocates a particular attitude toward that understanding. It is an attitude of disciplined openness. On the one hand, it leaves room for surprise. Instead of declaring in advance that hardware can contribute nothing beyond speed, it builds the experimental machinery to let the question be answered empirically, task by task and device by device. On the other hand, it insists that surprise be handled with rigor: anomalies are to be logged, reduced, measured, and modeled, not mythologized.

In this sense, substrate exploration is less about hunting for miracles and more about improving our habits of thought at the boundary between intelligence and matter. If, after extensive experimentation, no meaningful substrate-coupled advantages are found, we gain confidence that the digital abstraction is robust for current AI workloads, and that safety and certification can continue to focus primarily on models and data. If, instead, structured substrate effects are

uncovered and understood, they may lead to better hardware, better algorithms, and more realistic governance.

Either way, the process disciplines how we are surprised. It channels curiosity about the physical underpinnings of AI into a program of measurements and reductions, rather than speculation. And in doing so, it helps ensure that as intelligence and matter become more closely linked in our machines, our understanding of that link grows in step, anchored not in wishful thinking but in the slow, cumulative work of experiment and explanation.

References

1. Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(1), 230–265. londmathsoc.onlinelibrary.wiley.com+1

2. Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27(3–4), 379–423, 623–656. people.math.harvard.edu+1

3. Landauer, R. (1961). Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3), 183–191. ACM Digital Library+1

4. Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1), 5–48. galaxy.agh.edu.pl+1

5. Hennessy, J. L., & Patterson, D. A. (2017). *Computer Architecture: A Quantitative Approach* (6th ed.). Morgan Kaufmann. acs.pub.ro+1

6. Rabaey, J. M., Chandrakasan, A., & Nikolić, B. (2003). *Digital Integrated Circuits: A Design Perspective* (2nd ed.). Prentice Hall. kolegite.com+1

7. Mukherjee, S. (2008). *Architecture Design for Soft Errors*. Morgan Kaufmann. Amazon+1

8. Sze, V., Chen, Y.-H., Yang, T.-J., & Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12), 2295–2329. SemiWiki+1

9. Sze, V., Chen, Y.-H., Yang, T.-J., & Emer, J. S. (2020). *Efficient Processing of Deep Neural Networks*. Morgan & Claypool. eyeriss.mit.edu+1

10. Jouppi, N. P., Young, C., Patil, N., Patterson, D., et al. (2017). In-datacenter performance analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)* (pp. 1–12). ACM Digital Library+1

11. Indiveri, G., & Horiuchi, T. K. (2011). Frontiers in neuromorphic engineering. *Frontiers in Neuroscience*, 5, 118. Frontiers+1

12. Thakur, C. S., Molin, J.-L., Cauwenberghs, G., Indiveri, G., Kumar, K., Pfeiffer, M., … Qiao, N. (2018). Large-scale neuromorphic spiking array processors: A review. *Frontiers in Neuroscience*, 12, 891. PubMed

13. Jaeger, H. (2001). *The "echo state" approach to analysing and training recurrent neural networks*. GMD Report 148, German National Research Center for Information Technology. Semantic Scholar+1

14. Maass, W., Natschläger, T., & Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11), 2531–2560. PubMed+1

15. Tanaka, G., Yamane, T., Héroux, J.-B., Nakane, R., Kanazawa, N., Takeda, S., … Nakajima, K. (2019). Recent advances in physical reservoir computing: A review. *Neural Networks*, 115, 100–123. ACM Digital Library+1

16. Stepney, S. (2024). Physical reservoir computing: A tutorial. *Natural Computing*. (Early access). SpringerLink

17. Carroll, T. L. (2021). Optimizing reservoir computers for signal classification. *Frontiers in Physiology*, 12, 685121. Frontiers

18. Nobukawa, S., et al. (2025). Deep neural network architectures and reservoir computing. *Frontiers in Computational Neuroscience*, 19, 685121 (article ID as cited in preprint/early access). PMC

19. Rabaey, J. M. (2003). *Digital Integrated Circuits: A Design Perspective* (2nd ed.). Prentice Hall. Jan M. Rabaey+1

20. Landauer, R. (2000). Irreversibility and heat generation in the computing process (reprint). *IBM Journal of Research and Development*, 44(1/2), 261–269. sites.pitt.edu+1

The author has published ~ 4,600 AI-assisted papers at: https://www.researchgate.net/profile/Douglas-Youvan/research . Google Scholar has indexed these preprints, and if you want a particular reference, the AI mode of a Google search (Gemini) has efficiently catalogued these preprints.